​Text

Description automatically generated

LOG3000 – INGÉNIERIE LOGICIELLE

**TP4**

Présenté à:

Mohammad Hamdaqa

Écrit par :

Cassy Charles (1947025)

Samy Checklat (1937812)

Remis le 17 Avril 2022

# 4.2. Questions

## **Q1. Dans chaque stratégie de déploiement ci-dessous, donnez la stratégie de déploiement appropriée qu'elle représente et un exemple concret.**

**Cas 1 : Stratégie de déploiement en continu**

Dans ce cas, le déploiement en continu est utilisé. Un déploiement continu remplace lentement les instances de la version précédente d'une application par des instances de la nouvelle version de l'application. Pendant cette période, l’ancienne et la nouvelle version de l’application cohabitent ensemble sans toucher aux fonctionnalités ni affecter l’expérience utilisateur. Ce qu’on peut observer dans les figures données.

**Exemple concret**:

La mise à jour d’une application. Lorsque le déploiement d’une application se fait dans les conteneurs, la mise à jour se fait d’un conteneur à un aitre. De cette manière, la dernière image du fournisseur d’applications est téléchargée et modifiée sur chaque conteneur. Si des problèmes de compatibilité se présentent, le conteneur peut être recréé avec l’ancienne image. De cette manière, les deux versions, anciennes et nouvelles coexistent ensemble jusqu’à la mise à jour complète.

**Cas 2 :**

i) **Stratégie Blue-Green:** Dans cette image, la couleur bleue représente l’environnement de production. C’est cet environnement qui reçoit le trafic des utilisateurs. Ce trafic est géré à l’aide d’un “load balancer” qui permet qu’une partie de l’environnement de production ne soit pas surchargé. La partie verte du dessin représente un clone de l’environnement de production. En revanche, ce clone n’est pas actif. De plus, ces deux environnements partagent les mêmes configurations ainsi que la même base de données.

ii) **A/B Testing** Dans cette image, la couleur verte représente la nouvelle version de l’application. Cet environnement permet de tester les performances ainsi que les fonctionnalités de la nouvelle version de l’application. Une fois les tests validés, le trafic de l’application de l’environnement bleu est dirigé vers l’environnement vert. L’environnement vert devient donc le nouvel environnement de production.

**Exemple concret :**

Concernant cette stratégie, il est possible de penser au routage du trafic. En effet, pour cette stratégie de déploiement, il faut mettre à jours les DNS CNAMES pour les hôtes. Dans le cas contraire, il faut modifier les paramètres du “load balancer” afin d’apercevoir des changements immédiats. En exemple, certaines fonctionnalités, comme le drainage de connexions dans ELB

**Cas 3 : Stratégie Canari**

Pour cette stratégie, il est possible de remarquer que cette dernière ressemble à la stratégie “blue-green”. La différence vient du fait que la stratégie “Canari” utilise une seule étape tout en ayant une approche progressive.

**Exemple concret :**

Ce type de déploiement permet d’implémenter de nouvelles fonctionnalités de l’application dans une petite infrastructure de l’application. Lorsque cette fonctionnalité est approuvée, seul une petite partie des utilisateurs y accèdent. Cela permet de minimiser l’impact sur l’application.

**Q2. Quelle est la relation entre OpenShift et Kubernetes?**

Kubernetes et OpenShift présentent tous deux une architecture robuste et évolutive qui permet le développement, le déploiement et la gestion d'applications rapides et à grande échelle. Ils fonctionnent tous deux sous la licence Apache 2.0. Red Hat® OpenShift® est une distribution Kubernetes, un produit logiciel commercialisé dérivé d'un projet open source. Red Hat OpenShift et Kubernetes sont tous deux des logiciels d'orchestration de conteneurs, mais Red Hat OpenShift est conditionné en tant que plate-forme open source d'entreprise en aval, ce qui signifie qu'il a subi des tests supplémentaires et qu'il contient des fonctionnalités supplémentaires qui ne sont pas disponibles dans le projet open source Kubernetes.

**Q3. Dans vos propres mots, qu'est-ce que les Kubernetes et pourquoi est-il utile pour DevOps?**

Kubernetes est une plateforme open source qui automatise les opérations de conteneurs Linux. Elle élimine une grande partie des processus manuels impliqués dans le déploiement et la mise à l'échelle des applications conteneurisées, ce qui veut dire qu'on peut regrouper des groupes d'hôtes exécutant des conteneurs Linux, et Kubernetes aide à gérer facilement et efficacement ces conteneurs.

L'une des principales raisons d'utiliser Kubernetes pour DevOps est qu'il réduit la charge de travail. Kubernetes améliore la qualité des processus DevOps avec la cohérence de l'environnement de développement, de test et de déploiement, la prise en charge de plusieurs frameworks, l'intégration d'une nouvelle application prend moins de temps, l’amélioration de la productivité et de l'efficacité des développeurs. Kubernetes facilite aussi la livraison rapide des logiciels avec une meilleure conformité, améliore la collaboration et la transparence au sein des équipes chargées de livrer le logiciel et réduit efficacement les coûts de développement et les risques de sécurité.

**Q4. Expliquez la différence entre la mise à l'échelle horizontale et la mise à l'échelle verticale.**

La mise à l'échelle horizontale fait référence à l'ajout de plusieurs périphériques ou nœuds informatiques au système pour améliorer les performances, tandis que la mise à l'échelle verticale correspond à l'ajout de ressources supplémentaires à un seul périphérique informatique afin d'améliorer les performances. [1]

**Q5. Qu'est-ce qu'un pod? Expliquez comment ils fonctionnent.**

Les pods sont les plus petites unités déployables dans Kubernetes. Comme l'indique la documentation officielle : "Un pod est un groupe d'un ou plusieurs conteneurs, avec des ressources de stockage/réseau partagées, et une spécification sur la façon d'exécuter les conteneurs." Ainsi, dans les termes les plus simples possibles, un pod est le mécanisme qui permet d'activer un conteneur dans Kubernetes.

Les pods sont un produit des contrôleurs qui sont responsables de la gestion des opérations au sein du système Kubernetes. Les contrôleurs facilitent le déploiement, la réplication et l'état général des pods dans un cluster. Le contrôleur est le délégant de la réplication des pods si un pod échoue. Il existe trois principaux types de contrôleurs, à savoir les jobs, les déploiements et les statefulSets. Les travaux sont de courte durée, c'est-à-dire qu'ils n'existent que jusqu'à ce qu'un travail soit terminé. [2]

**Q6. Qu'est-ce que “l'autoguérison d'application”?**

Un système auto-réparateur peut découvrir des erreurs dans son fonctionnement et se modifier sans intervention humaine, ce qui lui permet de retrouver un meilleur état de fonctionnement. Dans les applications typiques, les problèmes sont documentés dans un "journal des exceptions" pour un examen ultérieur. La plupart des problèmes sont mineurs et peuvent être ignorés. Les problèmes graves peuvent nécessiter l'arrêt de l'application (par exemple, l'impossibilité de se connecter à une base de données qui a été mise hors ligne).

En revanche, les applications auto réparatrices intègrent des éléments de conception qui permettent de résoudre les problèmes. Par exemple, les applications qui utilisent Akka organisent les éléments dans une hiérarchie et attribuent les problèmes d'un acteur à son superviseur. Un grand nombre de bibliothèques et de frameworks de ce type facilitent les applications qui s'autoréparent par conception. [3]

**Q7. Quel est le but du routage?**

La première et principale partie du routage consiste à déterminer le chemin par lequel les paquets vont passer de l'hôte émetteur, ou d'origine, à l'hôte récepteur, ou de destination. La deuxième partie du processus consiste à demander aux routeurs de faire passer les paquets d'un segment successif, ou "saut", du chemin au suivant jusqu'à ce que les paquets arrivent à leur destination. Le routage permet aussi de convertir un nom d'URL en une adresse IP, d’assurer le transfert sécurisé de fichiers sur Internet et de transférer le trafic sur la base des adresses MAC.

**Q8. Dans l'interface du « portail d'apprentissage interactif », Sélectionnez deux scénarios: (1) Using the CLI to Manage Resource Objects (2) Transferring Files in and out of Containers Expliquez ce qui a été fait dans chaque scénario avec vos propres mots. Utilisez des captures d'écran pour appuyer votre réponse.**

1. **Using the CLI to Manage Resource Objects**

Avec la commande: oc login -u developer -p developer <https://api.crc.testing:6643> –insecure-skip-tls-verify=true, on se connecte en tant que développeur sur openshift. Et on crée un projet nommé my project : oc new-project myproject.

Text

Description automatically generated

Figure 1: Connexion et création d'un projet

Pour utiliser le cli pour gérer les objets ressources, nous devons d'abord déployer une application. Pour déployer notre application, nous exécutons la commande avec notre base de données PostGreSQL  : oc new-app postgresql-ephemeral -name database –param DATABASE\_SERVICE\_NAME = database –param… Nous surveillons alors le déploiement de l'application en exécutant : oc rollout status dc/database.

Text

Description automatically generated

Figure 2: déploiement de l'application

Nous pouvons voir le nom des pods correspondant aux conteneurs en cours d'exécution pour cette application en exécutant : oc get pods –selector name=database -o custom-columns. La liste de tous les nœuds du cluster pour obtenir des informations contenant la base de données est alors affichée. Et avec la commande oc rsh $POD, on accède localement aux outils présents sur le système et il est possible de les gérer.

Text

Description automatically generated

1. **Transferring Files in and out of Containers**

Avec la commande: oc login -u admin -p admin <https://api.crc.testing:6643> –insecure-skip-tls-verify=true, on se connecte en tant qu’admin sur openshift. Et on crée un projet nommé my project : oc new-project myproject.

Text

Description automatically generated

Figure 3: Authentification et création d'un projet

Pour démontrer le transfert de fichiers depuis et vers un conteneur en cours d'exécution, nous devons d'abord déployer une application. Pour déployer notre application, nous exécutons la commande : oc new-app openshiftkatacoda/blog-django-py --name blog. Pour y accéder depuis un navigateur Web, nous devons également l'exposer en créant une route : oc expose svc/blog. Nous pouvons également surveiller le déploiement de l'application en exécutant : oc rollout status dc/blog. Cette commande se terminera lorsque le déploiement sera terminé et que l'application web sera prête. Le résultat sera un conteneur en cours d'exécution. Nous pouvons voir le nom des pods correspondant aux conteneurs en cours d'exécution pour cette application en exécutant : oc get pods --selector app=blog.

Text

Description automatically generated

Figure 4: Déploiement d'une application et nom des pods correspondants aux conteneurs

Pour les commandes suivantes qui doivent interagir avec ce pod, on utilise le nom du pod comme argument. Comme dans la photo ci-dessus, dans ce cas, le pod est blog-5dc99d7545-qjkp. Pour créer un shell interactif dans le même conteneur que celui qui exécute l'application, on utilise la commande oc rsh, en lui fournissant la variable d'environnement contenant le nom du pod : oc rsh $POD. À partir du shell interactif, on peut voir quels fichiers existent dans le répertoire de l'application avec la commande : ls -las.Text

Description automatically generated

Figure 5: Fichiers existant dans le répertoire de l’application

Pour l'application utilisée, cela a créé un fichier de base de données comme ceci : 44 -rw-rw-r-- 1 1000040000 root 44032 Apr 17 01:35 db.sqlite3. Pour confirmer dans quel répertoire se trouve le fichier, à l'intérieur du conteneur, on a fait la commande : pwd. Ce qui affiche :/opt/app-root/src. Pour quitter le shell interactif et revenir à la machine locale, on a fait exit. Maintenant, pour copier des fichiers du conteneur vers la machine locale, on a utilisé la commande oc rsync. Pour copier un seul fichier du conteneur vers la machine locale, on a fait : oc rsync $POD:/opt/app-root/src/db.sqlite3 .

Text

Description automatically generated

Figure 6: Copie des fichiers du conteurs vers la machine locale

On reçoit aussi la liste incrémentielle des fichiers : db.sqlite3 sent 43 bytes, received 44,130 bytes… On vérifie encore le contenu du répertoire actuel en exécutant la commande :ls -las. On voit alors que la machine locale a maintenant une copie du fichier : 44 -rw-rw-r-- 1 1000040000 root 44032 Apr 17 01:35 db.sqlite3. On crée notre répertoire uploads et on copie le répertoire media depuis le conteneur de notre pod sur la machine locale avec : oc rsync $POD:/opt/app-root/src/media/. Uploads.

Maintenant pour le téléchargement de fichiers vers un conteneur :

Pour copier des fichiers de la machine locale vers le conteneur, on utilise encore la commande oc rsync. Contrairement à la copie du conteneur vers la machine locale, il n'y a pas de formulaire pour copier un seul fichier. Pour illustrer le processus de copie d'un seul fichier, on va inclure un fichier robots.txt dans un site web déployé. Tout d'abord, on crér un fichier robots.txt dans notre répertoire local qui contient : User-agent : \*, Disallow : /. Pour l'application web utilisée, il héberge des fichiers statiques provenant du sous-répertoire htdocs du code source de l'application. On télécharge alors le fichier robots.txt.

Text

Description automatically generated

Figure 7: Téléchargement du fichier web

Text

Description automatically generated

Figure 8: Copie des fichiers

Lorsqu’on copie des fichiers dans le conteneur, il est nécessaire que le répertoire dans lequel les fichiers sont copiés existe et qu'il soit accessible en écriture à l'utilisateur ou au groupe qui exécute le conteneur. Les permissions sur les répertoires et les fichiers doivent être définies dans le cadre du processus de construction de l'image. Dans la commande ci-dessus, l'option --no-perms est utilisée, car le répertoire cible du conteneur, bien qu'accessible en écriture par le groupe sous lequel le conteneur est exécuté, appartient à un utilisateur différent. Cela signifie que, bien que les fichiers puissent être ajoutés au répertoire, les permissions sur les répertoires existants ne peuvent pas être modifiées. L'option --no-perms indique à oc rsync de ne pas essayer de mettre à jour les permissions ; cela évite qu'il échoue et renvoie des erreurs. Maintenant que le fichier robots.txt est téléchargé, la demande de ce fichier va être lancée. Cela a fonctionné sans qu'il soit nécessaire de prendre d'autres mesures, car le serveur utilisé pour héberger les fichiers statiques détecte automatiquement la présence d'un nouveau fichier dans le répertoire. Pour copier le contenu complet d'un répertoire dans le répertoire htdocs du conteneur, on exécute: oc rsync images $POD:/opt/app-root/src/htdocs --no-perms.

Text

Description automatically generated

Figure 9: Vue du fichier Robots.txt

Graphical user interface, text

Description automatically generated

Figure 10: la liste incrémentielle des fichiers

Text

Description automatically generated

Figure 11: Closure de tous les processus

Text

Description automatically generated

Figure 12: Efface des pods

**Q9. Maintenant, vous devez implémenter le déploiement bleu-vert avec le code source disponible sur GitHub.**

La commande “oc new-project..” permet de créer un nouveau projet sur OpenShift. Ce projet porte le nom de “bluegreen”. En revanche le nom affiché sera “Blue Green” avec une description “Blue Green Deployments”

La commande “oc new-app" permet de générer un nouvel objet OpenShift Entreprise. Cet objet permet de construire l’application, la déployer ainsi que la rouler. Le nom du “pod” pour ce projet sera “blue” tout en ayant une stratégie source. L’objet est généré à partir du répertoire GitHub de Cassyie.

Text

Description automatically generated

Figure 13:Création du projet Blue Green sur OpenShift et du pod ”blue”

Cette commande permet de rendre le “pod” par des personnes externes. Une route est créée et sera accessible aux personnes externes à partir de route.route.openshift.io/bluegreen

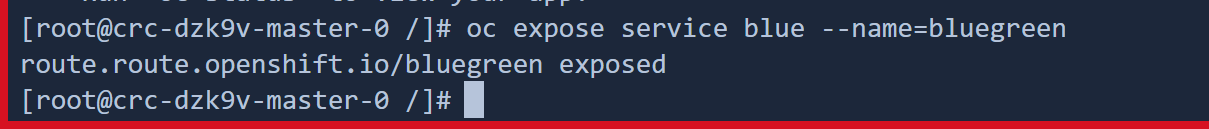


Figure 14: Création de la route accessible pour des personnes externes.

Afin de pouvoir apporter des modifications, nous avons cloner le répertoire du projet dans notre compte GitHub. Nous avons ensuite créé une branche portant le nom “green”. Nous avons ensuite push cette nouvelle branche dans le répertoire

Text

Description automatically generated

Figure 15: Ajout de la branche green dans le répertoire du projet

La commande “oc new-app" permet de générer un nouvel objet OpenShift Entreprise. Cet objet permet de construire l’application, la déployer ainsi que la rouler. Le nom du “pod” pour ce projet sera “green” tout en ayant une stratégie source. L’objet est généré à partir du répertoire GitHub de Cassyie.

Text

Description automatically generated

Figure 16: Création du pod ”green”

Cette commande permet de modifier la route initiale qui était associé au pod “blue” vers le pod “green”.

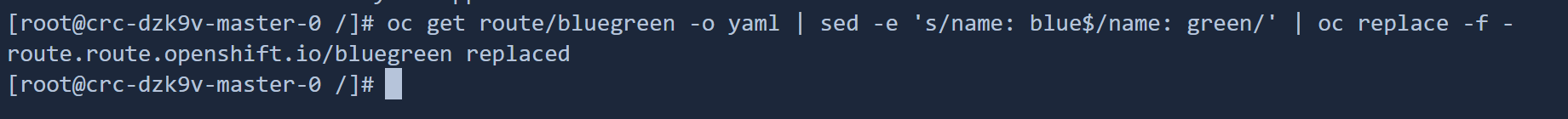


Figure 17: Changement du service associé à la route du pod ”blue” vers le pod ”green”

Cette commande permet de modifier la route initiale qui était associé au pod “green” vers le pod “blue”.

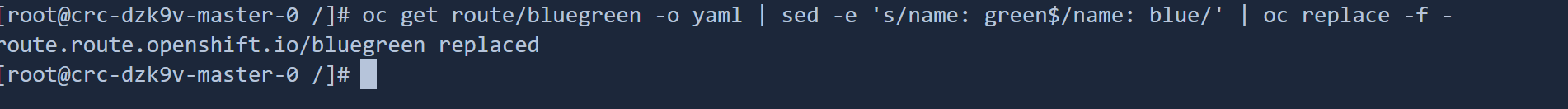


Figure 18: Changement du service associé à la route du pod ”green” vers le pod ”blue”

Graphical user interface, application

Description automatically generated

Figure 19: Affichage des pods fonctionnels

Graphical user interface, application

Description automatically generated

Figure 20: Affichage des pods fonctionnels dans l’interface de configuration

# 4.3. Question de rétroaction

**Nous travaillons à l’amélioration continue des travaux pratiques de LOG3000. Cette question peut être répondue très brièvement. Combien de temps avez-vous passé au travail pratique, en heures-personnes, en sachant que deux personnes travaillant pendant trois heures correspondent à six heures-personnes ? Est-ce que l'effort demandé pour ce laboratoire est adéquat ?**

Nous avons passé 8 heures-personnes sur ce travail pratique car la partie tutoriel nous a pris un peu de temps. L’effort demandé pour ce laboratoire est adéquat.

# Références

|  |  |
| --- | --- |
| [1] | «Quelle est la différence entre la mise à l'échelle horizontale et verticale,» Sawakinome, [En ligne]. Available: https://fr.sawakinome.com/articles/technology/what-is-the-difference-between-horizontal-and-vertical-scaling.html. |
| [2] | «How Kubernetes Pods Work,» 6 Fevrier 2021. [En ligne]. Available: https://www.section.io/engineering-education/how-kubernetes-pods-work/. |
| [3] | S. Ray, «What Are Self Healing Systems?,» 19 Fevrier 2020. [En ligne]. Available: https://medium.com/lansaar/what-are-self-healing-systems-42ac9dd0e0aa. |